Modeling Adversaries in a Logic for Security Protocol Analysis*

Joseph Y. Halpern
Cornell University
Ithaca, NY 14853
halpern@cs.cornell.edu

Riccardo Pucella
Northeastern University
Boston, MA 02115
riccardo@ccs.neu.edu

Abstract

Logics for security protocol analysis require the formalization of an adversary model that specifies the capabilities of adversaries. A common model is the Dolev-Yao model, which considers only adversaries that can compose and replay messages, and decipher them with known keys. The Dolev-Yao model is a useful abstraction, but it suffers from some drawbacks: it cannot handle the adversary knowing protocol-specific information, and it cannot handle probabilistic notions, such as the adversary attempting to guess the keys. We show how we can analyze security protocols under different adversary models by using a logic with a notion of algorithmic knowledge. Roughly speaking, adversaries are assumed to use algorithms to compute their knowledge; adversary capabilities are captured by suitable restrictions on the algorithms used. We show how we can model the standard Dolev-Yao adversary in this setting, and how we can capture more general capabilities including protocol-specific knowledge and guesses.

1 Introduction

Many formal methods for the analysis of security protocols rely on specialized logics to rigorously prove properties of the protocols they study.[1] Those logics provide constructs for expressing the basic notions involved in security protocols, such as secrecy, recency, and message composition, as well as providing means (either implicitly or explicitly) for describing the evolution of the knowledge or belief of the principals as the protocol progresses. Every such logic aims at proving security in the presence of hostile adversaries. To analyze the effect of adversaries, a security logic specifies (again, either implicitly or explicitly) an *adversary model*, that is, a description of the capabilities of adversaries. Almost all existing logics are based on a Dolev-Yao adversary model [Dolev and Yao 1983]. Succinctly, a Dolev-Yao adversary can compose messages, replay them, or decipher them if she knows the right keys, but cannot otherwise "crack" encrypted messages.

*A preliminary version of this paper appeared in the *Proceedings of the Workshop on Formal Aspects of Security*, LNCS 2629, pp. 115-132, 2002. This work was done while the second author was at Cornell University.

[1]Here, we take a very general view of logic, to encompass formal methods where the specification language is implicit, or where the properties to be checked are fixed, such as Casper [Lowe 1998], Cryptyc [Gordon and Jeffrey 2003], or the NRL Protocol Analyzer [Meadows 1996].

The Dolev-Yao adversary is a useful abstraction, in that it allows reasoning about protocols without worrying about the actual encryption scheme being used. It also has the advantage of being restricted enough that interesting theorems can be proved with respect to security. However, in many ways, the Dolev-Yao model is too restrictive. For example, it does not consider the information an adversary may infer from properties of messages and knowledge about the protocol that is being used. To give an extreme example, consider what we will call the *Duck-Duck-Goose* protocol: an agent has an n-bit key and, according to her protocol, sends the bits that make up its key one by one. Of course, after intercepting these messages, an adversary will know the key. However, there is no way for security logics based on a Dolev-Yao adversary to argue that, at this point, the adversary knows the key. Another limitation of the Dolev-Yao adversary is that it does not easily capture probabilistic arguments. After all, the adversary can always be lucky and just *guess* the appropriate key to use, irrespective of the strength of the encryption scheme.

The importance of being able to reason about adversaries with capabilities beyond those of a Dolev-Yao adversary is made clear when we look at the subtle interactions between the cryptographic protocol and the encryption scheme. It is known that various protocols that are secure with respect to a Dolev-Yao adversary can be broken when implemented using encryption schemes with specific properties [Moore 1988], such as encryption systems with encryption cycles [Abadi and Rogaway 2002] and ones that use exclusive-or [Ryan and Schneider 1998]. A more refined logic for reasoning about security protocols will have to be able to handle adversaries more general than the Dolev-Yao adversary.

Because they effectively build in the adversary model, existing formal methods for analyzing protocols are not able to reason directly about the effect of running a protocol against adversaries with properties other than those built in. The problem is even worse when it is not clear exactly what assumptions are implicitly being made about the adversary. One obvious assumption that needs to be made clear is whether the adversary is an insider in the system or an outsider. Lowe's [1995] well-known man-in-the-middle attack against the Needham-Schroeder [1978] protocol highlights this issue. Until then, the Needham-Schroeder protocol had been analyzed under the assumption that the adversary had complete control of the network, and could inject intercept and inject arbitrary messages (up to the Dolev-Yao capabilities) into the protocol runs. However, the adversary was always assumed to be an outsider, not being able to directly interact with the protocol principals as himself. Lowe showed that if the adversary is allowed to be an *insider* of the system, that is, appear to the other principals as a bona fide protocol participant, then the Needham-Schroeder protocol does not guarantee the authentication properties it is meant to guarantee.

In this paper, we introduce a logic for reasoning about security protocols that allows us to model adversaries explicitly and naturally. The idea is to model the adversary in terms of what the adversary knows. This approach has some significant advantages. Logics of knowledge [Fagin, Halpern, Moses, and Vardi 1995] have been shown to provide powerful methods for reasoning about trace-based executions of protocols. They can be given semantics that is tied directly to protocol execution, thus avoiding problems of having to analyze an idealized form of the protocol, as is required, for example, in BAN logic [Burrows, Abadi, and Needham 1990]. A straightforward application of logics of knowledge allows us to conclude that in the Duck-Duck-Goose protocol, the adversary knows the key. Logics of knowledge

can also be extended with probabilities [Fagin and Halpern 1994; Halpern and Tuttle 1993] so as to be able to deal with probabilistic phenomena. Unfortunately, traditional logics of knowledge suffer from a well-known problem known as the *logical omniscience* problem: an agent knows all tautologies and all the logical consequences of her knowledge. The reasoning that allows an agent to infer properties of the protocol also allows an attacker to deduce properties that cannot be computed by realistic attackers in any reasonable amount of time.

To avoid the logical omniscience problem, we use the notion of *algorithmic knowledge* [Fagin, Halpern, Moses, and Vardi 1995, Chapter 10 and 11]. Roughly speaking, we assume that agents (including adversaries) have "knowledge algorithms" that they use to compute what they know. The capabilities of the adversary are captured by its algorithm. Hence, Dolev-Yao capabilities can be provided by using a knowledge algorithm that can only compose messages or attempt to decipher using known keys. By changing the algorithm, we can extend the capabilities of the adversary so that it can attempt to crack the encryption scheme by factoring (in the case of RSA), using differential cryptanalysis (in the case of DES), or just by guessing keys, along the lines of a model due to Lowe [2002]. Moreover, our framework can also handle the case of a principal sending the bits of its key, by providing the adversary's algorithm with a way to check whether this is indeed what is happening. By explicitly using algorithms, we can therefore analyze the effect of bounding the resources of the adversary, and thus make progress toward bridging the gap between the analysis of cryptographic protocols and more computational accounts of cryptography. (See [Abadi and Rogaway 2002] and the references therein for a discussion on work bridging this gap.) Note that we need both traditional knowledge and algorithmic knowledge in our analysis. Traditional knowledge is used to model a principal's beliefs about what can happen in the protocol; algorithmic knowledge is used to model the adversary's computational limitations (for example, the fact that it cannot factor).

The focus of this work is on developing a general and expressive framework for modeling and reasoning about security protocols, in which a wide class of adversaries can be represented naturally. Therefore, we emphasize the expressiveness and representability aspects of the framework, rather than studying the kind of security properties that are useful in such a setting or developing techniques for proving that properties hold in the framework. These are all relevant questions that need to be pursued once the framework proves useful as a specification language.

The rest of the paper is organized as follows. In Section 3, we define our model for protocol analysis and our logic for reasoning about implicit and explicit knowledge, based on the well-understood multiagent system framework. In Section 4, we show how to model different adversaries from the literature. These adversaries are passive, in that they eavesdrop on the communication but do not attempt to interact with the principals of the system. In Section 4.2, we show how the framework can accommodate active adversaries, that is, adversaries that can actively interact with the principals by intercepting, forwarding, and replacing messages. We discuss related work in Section 5.

2 Modeling Security Protocols

In this section, we review the multiagent system framework of Fagin et al. [1995, Chapters 4 and 5], and show it can be tailored to represent security protocols.

2.1 Multiagent Systems

The multiagent systems framework provides a model for knowledge that has the advantage of also providing a discipline for modeling executions of protocols. A multiagent system consists of n agents, each of which is in some local state at a given point in time. We assume that an agent's local state encapsulates all the information to which the agent has access. In the security setting, the local state of an agent might include some initial information regarding keys, the messages she has sent and received, and perhaps the reading of a clock. In a poker game, a player's local state might consist of the cards he currently holds, the bets made by other players, any other cards he has seen, and any information he may have about the strategies of the other players (for example, Bob may know that Alice likes to bluff, while Charlie tends to bet conservatively). The basic framework makes no assumptions about the precise nature of the local state.

We can then view the whole system as being in some global state, which is a tuple consisting of each agent' local state, together with the state of the environment, where the environment consists of everything that is relevant to the system that is not contained in the state of the agents. Thus, a global state has the form $(s_e, s_1, \ldots, s_n)$, where s_e is the state of the environment and s_i is agent i's state, for $i = 1, \ldots, n$. The actual form of the agents' local states and the environment's state depends on the application.

A system is not a static entity. To capture its dynamic aspects, we define a run to be a function from time to global states. Intuitively, a run is a complete description of what happens over time in one possible execution of the system. A *point* is a pair (r, m) consisting of a run r and a time m. For simplicity, we take time to range over the natural numbers in the remainder of this discussion. At a point (r, m), the system is in some global state $r(m)$. If $r(m) = (s_e, s_1, \ldots, s_n)$, then we take $r_i(m)$ to be s_i, agent i's local state at the point (r, m). We formally define a system $\mathcal{R}$ to consist of a set of runs (or executions). It is relatively straightforward to model security protocols as systems. Note that the adversary in a security protocol can be modeled as just another agent. The adversary's information at a point in a run can be modeled by his local state.

2.2 Specializing to Security

The multiagent systems framework is quite general. We have a particular application in mind, namely reasoning about security protocols, especially authentication protocols. We now specialize the framework in a way appropriate for reasoning about security protocols.

Since the vast majority of security protocols studied in the literature are message-based, a natural class of multiagent systems to consider is that of *message passing systems* [Fagin, Halpern, Moses, and Vardi 1995]. Let $\mathcal{M}$ be a fixed set of messages. A *history* for agent i (over $\mathcal{M}$) is a sequence of elements of the form $\mathsf{send}(j, \mathsf{m})$ and $\mathsf{recv}(\mathsf{m})$, where $\mathsf{m} \in \mathcal{M}$. We think of $\mathsf{send}(j, \mathsf{m})$ as representing the event "message m is sent to j" and $\mathsf{recv}(\mathsf{m})$ as representing the event "message m is received.". (We can also allow events corresponding to internal actions; however, since internal actions do not play any role in this paper, we choose to ignore those events for the time being.) Intuitively, i's history at (r, m) consists of i's initial state, which we take to be the empty sequence, followed by the sequence describing i's actions up to time m. If i performs no actions in round m, then her history at (r, m) is the same as its history at $(r, m-1)$. In such a message-passing system, we speak of $\mathsf{send}(j, \mathsf{m})$

and $\mathsf{recv}(\mathsf{m})$ as *events*. For an agent i, let $r_i(m)$ be agent i's history in (r, m). We say that an event e *occurs in* i*'s history in round* $m+1$ *of run* r if e is in (the sequence) $r_i(m+1)$ but not in $r_i(m)$.

In a message-passing system, the agent's local state at any point is her history. Of course, if h is the history of agent i at the point (r, m), then we want it to be the case that h describes what happened in r up to time m from i's point of view. To do this, we need to impose some consistency conditions on global states. In particular, we want to ensure that message histories do not shrink over time, and that every message received in round m corresponds to a message that was sent at some earlier round.

Given a set $\mathcal{M}$ of messages, we define a *message-passing system* (over $\mathcal{M}$) to be a system satisfying the following constraints at all points (r, m) for each agent i:

MP1. $r_i(m)$ is a history over $\mathcal{M}$.

MP2. For every event $\mathsf{recv}(\mathsf{m})$ in $r_i(m)$ there exists a corresponding event $\mathsf{send}(j, \mathsf{m})$ in $r_j(m)$, for some j.

MP3. $r_i(0)$ is the empty sequence and $r_i(m+1)$ is either identical to $r_i(m)$ or the result of appending one event to $r_i(m)$.

MP1 says that an agent's local state is her history, MP2 guarantees that every message received at round m corresponds to one that was sent earlier, and MP3 guarantees that histories do not shrink.

A *security system* is a message passing system where the message space has a structure suitable for the interpretation of security protocols. Therefore, a security system assumes a set $\mathcal{P}$ of plaintexts, as well as a set $\mathcal{K}$ of keys. For every key $\mathsf{k} \in \mathcal{K}$, there corresponds an inverse key $\mathsf{k}^{-1} \in \mathcal{K}$ (which could be equal to k). A encryption scheme $\mathcal{C}$ over $\mathcal{P}$ and $\mathcal{K}$ is the closure $\mathcal{M}$ of $\mathcal{P}$ and $\mathcal{K}$ under a concatenation operation $conc : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$, decomposition operators $first : \mathcal{M} \to \mathcal{M}$ and $second : \mathcal{M} \to \mathcal{M}$, an encryption operation $encr : \mathcal{M} \times \mathcal{K} \to \mathcal{M}$, and a decryption operation $decr : \mathcal{M} \times \mathcal{K} \to \mathcal{M}$, subject to the constraints:

$$\begin{aligned} first(conc(\mathsf{m}_1, \mathsf{m}_2)) &= \mathsf{m}_1 \\ second(conc(\mathsf{m}_1, \mathsf{m}_2)) &= \mathsf{m}_2 \\ decr(encr(\mathsf{m}, \mathsf{k}), \mathsf{k}^{-1}) &= \mathsf{m}. \end{aligned}$$

In other words, decryption an encrypted message with the inverse of the key used to encrypt the message yields the original message. (For simplicity, we restrict ourselves to nonprobabilistic encryption schemes in this paper.) We often write $\mathsf{m}_1 \cdot \mathsf{m}_2$ for $conc(\mathsf{m}_1, \mathsf{m}_2)$ and $\{\!|\mathsf{m}|\!\}_\mathsf{k}$ for $encr(\mathsf{m}, \mathsf{k})$. There is no difficulty in adding more operations to the encryption schemes, for instance, to model hashes, signatures, or the ability to take the exclusive-or of two terms. We make no assumption in the general case as to the properties of encryption. Thus, for instance, most concrete encryption schemes allow collisions, that is, $\{\!|m_1|\!\}_{k_1} = \{\!|m_2|\!\}_{k_2}$ without $m_1 = m_2$ and $k_1 = k_2$. (In contrast, most security protocol analyses assume that there are no properties of encryption schemes beyond those specified above; this is part of the Dolev-Yao adversary model, which we examine in more detail in Section 4.1.1.)

Define $\sqsubseteq$ on $\mathcal{M}$ as the smallest relation satisfying the following constraints:

(1) $\mathsf{m} \sqsubseteq \mathsf{m}$

(2) if $\mathsf{m} \sqsubseteq \mathsf{m}_1$, then $\mathsf{m} \sqsubseteq \mathsf{m}_1 \cdot \mathsf{m}_2$

(3) if $\mathsf{m} \sqsubseteq \mathsf{m}_2$, then $\mathsf{m} \sqsubseteq \mathsf{m}_1 \cdot \mathsf{m}_2$

(4) if $\mathsf{m} \sqsubseteq \mathsf{m}_1$, then $\mathsf{m} \sqsubseteq \{\!|\mathsf{m}_1|\!\}_\mathsf{k}$.

Intuitively, $\mathsf{m}_1 \sqsubseteq \mathsf{m}_2$ if m_1 *could* be used in the construction of m_2. For example, if $\mathsf{m} = \{\!|\mathsf{m}_1|\!\}_\mathsf{k} = \{\!|\mathsf{m}_2|\!\}_\mathsf{k}$, then both $\mathsf{m}_1 \sqsubseteq \mathsf{m}$ and $\mathsf{m}_2 \sqsubseteq \mathsf{m}$. Therefore, if we want to establish that $\mathsf{m}_1 \sqsubseteq \mathsf{m}_2$ for a given m_1 and m_2, then we have to look at all the possible ways in which m_2 can be taken apart, either by concatenation or encryption, to finally decide if m_1 can be derived from m_2. Clearly, if encryption does result in collisions, there is a single way in which m_2 can be taken apart.

To analyze a particular security protocol, we first derive the multiagent system corresponding to the protocol, using the approach of Fagin et al. [1995, Chapter 5]. Intuitively, this multiagent system contains a run for every possible execution of the protocol, for instance, for every possible key used by the principals, subject to the restrictions above (such as MP1–MP3).

Formally, a protocol for agent i is a function from her local state to the set of actions that she can perform at that state. For ease of exposition, the only actions we consider here are those of sending messages (although we could easily incorporate other actions, such as choosing keys, or tossing coins to randomize protocols). A *joint protocol* $P = (P_e, P_1, \ldots, P_n)$, consisting of a protocol for each of the agents (including a protocol for the environment), associates with each global state a set of possible *joint actions* (i.e., tuples of actions) in the obvious way. Joint actions transform global states. To capture their effect, we associate with every joint action a a function $\tau(\mathsf{a})$ from global states to global states. This function captures, for instance, the fact that a message sent by an agent will be received by another agent, and so on. Given a context consisting of a set of initial global states, an interpretation τ for the joint actions, and a protocol P_e for the environment, we can generate a system corresponding to the joint protocol P in a straightforward way. Intuitively, the system consists of all the runs r that could have been generated by the joint protocol P, that is, for all m, $r(m+1)$ is the result of applying $\tau(\mathsf{a})$ to $r(m)$, where a is a joint action that could have been performed according to the joint protocol P to $r(m)$.[2]

3 A Logic for Security Properties

The aim is to be able to reason about properties of security systems as defined in the last section, including properties involving the knowledge of agents in the system. To formalize this type of reasoning, we first need a language. The logic of algorithmic knowledge [Fagin, Halpern, Moses, and Vardi 1995, Chapters 10 and 11] provides such a framework. It extends the classical logic of knowledge by adding algorithmic knowledge operators.

[2]It is also possible to represent a protocol other ways, such as in terms of *strand spaces* [Thayer, Herzog, and Guttman 1999]. Whichever representation is used, it should be possible to get a system corresponding to the protocol. For example, Halpern and Pucella [2003] show how to get a system from a strand space representation. For the purposes of this paper, the precise mechanism used to derive the multiagent system is not central, although it is an important issue for the development of formal tools for analyzing protocols.

The syntax of the logic $\mathcal{L}_n^{\mathrm{KX}}$ for algorithmic knowledge is straightforward. Starting with a set Φ_0 of primitive propositions, which we can think of as describing basic facts about the system, such as "the key is k" or "agent A sent the message m to B", formulas of $\mathcal{L}_n^{\mathrm{KX}}(\Phi_0)$ are formed by closing off under negation, conjunction, and the modal operators $K_1, \ldots, K_n$ and $X_1, \ldots, X_n$.

The formula $K_i\varphi$ is read as "agent i (implicitly) knows the fact φ", while $X_i\varphi$ is read as "agent i explicitly knows fact φ". In fact, we will read $X_i\varphi$ as "agent i can compute fact φ". This reading will be made precise when we discuss the semantics of the logic. As usual, we take $\varphi \vee \psi$ to be an abbreviation for $\neg(\neg\varphi \wedge \neg\psi)$ and $\varphi \Rightarrow \psi$ to be an abbreviation for $\neg\varphi \vee \psi$.

The standard models for this logic are based on the idea of possible worlds and Kripke structures [Kripke 1963]. Formally, a Kripke structure M is a tuple $(S, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n)$, where S is a set of states or possible worlds, π is an interpretation which associates with each state in S a truth assignment to the primitive propositions (i.e., $\pi(s)(p) \in \{\mathbf{true}, \mathbf{false}\}$ for each state $s \in S$ and each primitive proposition p), and $\mathcal{K}_i$ is an equivalence relation on S (recall that an equivalence relation is a binary relation which is reflexive, symmetric, and transitive). $\mathcal{K}_i$ is agent i's possibility relation. Intuitively, $(s,t) \in \mathcal{K}_i$ if agent i cannot distinguish state s from state t (so that if s is the actual state of the world, agent i would consider t a possible state of the world).

A system can be viewed as a Kripke structure, once we add a function π telling us how to assign truth values to the primitive propositions. An *interpreted system* $\mathcal{I}$ consists of a pair $(\mathcal{R}, \pi)$, where $\mathcal{R}$ is a system and π is an interpretation for the propositions in Φ that assigns truth values to the primitive propositions at the global states. Thus, for every $p \in \Phi$ and global state s that arises in $\mathcal{R}$, we have $\pi(s)(p) \in \{\mathbf{true}, \mathbf{false}\}$. Of course, π also induces an interpretation over the points of $\mathcal{R}$; simply take $\pi(r,m)$ to be $\pi(r(m))$. We refer to the points of the system $\mathcal{R}$ as points of the interpreted system $\mathcal{I}$.

The interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$ can be made into a Kripke structure by taking the possible worlds to be the points of $\mathcal{R}$, and by defining $\mathcal{K}_i$ so that $((r,m),(r',m')) \in \mathcal{K}_i$ if $r_i(m) = r'_i(m')$. Clearly $\mathcal{K}_i$ is an equivalence relation on points. Intuitively, agent i considers a point (r',m') possible at a point (r,m) if i has the same local state at both points. Thus, the agents' knowledge is completely determined by their local states.

To account for X_i, we provide each agent with a knowledge algorithm that he uses to compute his knowledge. We will refer to $X_i\varphi$ as *algorithmic knowledge*. An *interpreted algorithmic knowledge system* has the form $(\mathcal{R}, \pi, \mathtt{A}_1, \ldots, \mathtt{A}_n)$, where $(\mathcal{R}, \pi)$ is an interpreted system and $\mathtt{A}_i$ is the knowledge algorithm of agent i. In local state ℓ, the agent computes whether he knows φ by applying the knowledge algorithm $\mathtt{A}$ to input (φ, ℓ). The output is either "Yes", in which case the agent knows φ to be true, "No", in which case the agent does *not* know φ to be true, or "?", which intuitively says that the algorithm has insufficient resources to compute the answer. It is the last clause that allows us to deal with resource-bounded reasoners.

We define what it means for a formula φ to be true (or satisfied) at a point (r,m) in an interpreted system $\mathcal{I}$, written $(\mathcal{I}, r, m) \models \varphi$, inductively as follows:

$(\mathcal{I}, r, m) \models p$ if $\pi(r,m)(p) = \mathbf{true}$

$(\mathcal{I}, r, m) \models \neg\varphi$ if $(\mathcal{I}, r, m) \not\models \varphi$

$(\mathcal{I}, r, m) \models \varphi \wedge \psi$ if $(\mathcal{I}, r, m) \models \varphi$ and $(\mathcal{I}, r, m) \models \psi$

$(\mathcal{I}, r, m) \models K_i\varphi$ if $(\mathcal{I}, r, m) \models \varphi$ for all (r', m') such that $r_i(m) = r'_i(m')$

$(\mathcal{I}, r, m) \models X_i\varphi$ if $\mathtt{A}_i(\varphi, r_i(m)) =$ "Yes".

The first clause shows how we use the π to define the semantics of the primitive propositions. The next two clauses, which define the semantics of $\neg$ and $\wedge$, are the standard clauses from propositional logic. The fourth clause is designed to capture the intuition that agent i knows φ exactly if φ is true in all the worlds that i thinks are possible. The last clause captures the fact that explicit knowledge is determined using the knowledge algorithm of the agent.

As we pointed out, we think of K_i as representing *implicit knowledge*, facts that the agent implicitly knows, given its information, while X_i represents *explicit knowledge*, facts whose truth the agent can compute explicitly. As is well known, implicit knowledge suffers from the logical omniscience problem; agents implicitly know all valid formulas and agents implicitly know all the logical consequences of their knowledge (that is, $(K_\varphi \wedge K_i(\varphi \Rightarrow \psi)) \Rightarrow K_i\psi$ is valid). Explicit knowledge does not have that problem. Note that, as defined, there is no necessary connection between $X_i\varphi$ and $K_i\varphi$. An algorithm could very well claim that agent i knows φ (i.e., output "Yes") whenever it chooses to, including at points where $K_i\varphi$ does not hold. Although algorithms that make mistakes are common, we are often interested in knowledge algorithms that are correct. A knowledge algorithm is *sound* for agent i in the system $\mathcal{I}$ if for all points (r, m) of $\mathcal{I}$ and formulas φ, $\mathtt{A}(\varphi, r_i(m)) =$ "Yes" implies $(\mathcal{I}, r, m) \models K_i\varphi$, and $\mathtt{A}(\varphi, r_i(m)) =$ "No" implies $(\mathcal{I}, r, m) \models \neg K_i\varphi$. Thus, a knowledge algorithm is sound if its answers are always correct.

To reason about security protocols, we use the following set Φ_0^{S} of primitive propositions:

- $\mathsf{send}_i(\mathsf{m})$: agent i sent message m;
- $\mathsf{recv}_i(\mathsf{m})$: agent i received message m;
- $\mathsf{has}_i(\mathsf{m})$: agent i has message m.

Intuitively, $\mathsf{send}_i(\mathsf{m})$ is true when agent i has sent message m at some point, and $\mathsf{recv}_i(\mathsf{m})$ is true when agent i has received message m at some point. Agent i *has* a submessage m_1 at a point (r, m), written $\mathsf{has}_i(\mathsf{m}_1)$, if there exists a message $\mathsf{m}_2 \in \mathcal{M}$ such that $\mathsf{recv}(\mathsf{m}_2)$ is in $r_i(m)$, the local state of agent i, and $\mathsf{m}_1 \sqsubseteq \mathsf{m}_2$. Note that the has_i predicate is not restricted by issues of encryption. If $\mathsf{has}_i(\{\!|\mathsf{m}|\!\}_\mathsf{k})$ holds, then so does $\mathsf{has}_i(\mathsf{m})$, whether or not agent i knows the key k^{-1}. Intuitively, the has_i predicate characterizes the messages that agent i has implicitly in her possession.

An *interpreted algorithmic knowledge security system* is simply an interpreted algorithmic knowledge system $\mathcal{I} = (\mathcal{R}, \pi, \mathtt{A}_1, \ldots, \mathtt{A}_n)$, where $\mathcal{R}$ is a security system, the set Φ_0 of primitive propositions includes Φ_0^{S}, and π is an *acceptable* interpretation, that is, it gives the following fixed interpretation to the primitive propositions in Φ_0^{S}:

- $\pi_{\mathcal{R}}^{\mathrm{S}}(r, m)(\mathsf{send}_i(\mathsf{m})) = \mathbf{true}$ if and only if there exists j such that $\mathsf{send}(j, \mathsf{m}) \in r_i(m)$
- $\pi_{\mathcal{R}}^{\mathrm{S}}(r, m)(\mathsf{recv}_i(\mathsf{m})) = \mathbf{true}$ if and only if $\mathsf{recv}(\mathsf{m}) \in r_i(m)$

- $\pi^{\mathrm{S}}_{\mathcal{R}}(r,m)(\mathsf{has}_i(\mathsf{m})) = \mathbf{true}$ if and only if there exists m' such that $\mathsf{m} \sqsubseteq \mathsf{m}'$ and $\mathsf{recv}(\mathsf{m}') \in r_i(m)$.

This language can easily express the type of confidentiality (or secrecy) properties that we focus on here. Intuitively, we want to guarantee that throughout a protocol interaction, the adversary does not know a particular message. Confidentiality properties are stated naturally in terms of knowledge, for example, "agent 1 knows that the key k is a key known only to agent 2 and herself". Confidentiality properties are well studied, and central to most of the approaches to reasoning about security protocols.[3] Higher-level security properties, such as authentication properties, can often be established via confidentiality properties. See [Syverson and Cervesato 2001] for more details.

To illustrate some of the issues involved, consider an authentication protocol such as the Needham-Schroeder-Lowe protocol [Lowe 1995]. A simplified version of the protocol is characterized by the following message exchange between two agents A and B:

$$\begin{aligned} A &\rightarrow B : \{\!|n_A, A|\!\}_{k_B} \\ B &\rightarrow A : \{\!|n_A, n_B, B|\!\}_{k_A} \\ A &\rightarrow B : \{\!|n_B|\!\}_{k_B}. \end{aligned}$$

An authentication property of this protocol can be expressed informally as follows: under suitable assumptions on the keys known to the adversary and the fact that B is running his part of the protocol, A knows that n_A and n_B are kept confidential between her and B.[4] ¿From this, she knows that she is interacting with B, because she has received a message containing n_A, which only B could have produced. Similarly, A also knows that when B receives her message, B will know that he is interacting with A, because only A knows the nonce n_B which is part of the last message. Similar reasoning can be applied to B. This argument relies on the confidentiality of the nonces n_a and n_b. Using knowledge, this is simply the fact that no agents other than A and B know $\mathsf{has}_i(n_A)$ or $\mathsf{has}_i(n_B)$.

The fact that the implicit knowledge operator suffers from logical omniscience is particularly relevant here. At every point where an adversary i intercepts a message $\{\!|n_A, n_B, B|\!\}_{k_A}$, $K_i\mathsf{has}_i(n_A)$ is true (since $n_A \sqsubseteq \{\!|n_A, n_B, B|\!\}_{k_A}$), and hence the adversary knows that he has seen the nonce n_A, irrespective of whether he knows the decryption key corresponding to k_A). This shows that the implicit knowledge operator does not capture important aspects of reasoning about security. The adversary having the implicit knowledge that n_A is part of the message does not suffice, in general, for the adversary to *explicitly* know that n_A is part of the message. Intuitively, the adversary may not have the capabilities to realize he has seen n_A.

A more reasonable interpretation of confidentiality in this particular setting is $X_i\mathsf{has}_i(n_A)$: the adversary does not explicitly know, that is, cannot compute, whether he has seen the nonce n_A. Most logics of security introduce special primitives to capture the fact that the adversary can see a message m encrypted with key k only if he has access to the key k.

[3] A general definition of secrecy in terms of knowledge is presented by Halpern and O'Neill [2002] in the context of information flow, a setting that does not take into account cryptography.

[4] It may be more reasonable to talk about *belief* rather than *knowledge* that n_A and n_B are kept confidential. For simplicity, we talk about knowledge in this paper. Since most representations of belief suffer from logical omniscience, what we say applies to belief as well as knowledge.

Doing this hardwires the capabilities of the adversary into the semantics. Changing these capabilities requires changing the semantics. In our case, we simply need to supply the appropriate knowledge algorithm to the adversary, capturing his capabilities. In the following section, we examine in more detail the kind of knowledge algorithms that correspond to interesting capabilities.

4 Modeling Adversaries

As we showed in the last two sections, interpreted algorithmic knowledge security systems can be used to provide a foundation for representing security protocols, and support a logic for writing properties based on knowledge, both traditional (implicit) and algorithmic (explicit). For the purposes of analyzing security protocols, we use traditional knowledge to model a principal's beliefs about what can happen in the protocol, while we use algorithmic knowledge to model the adversary's capabilities, possibly resource-bounded. To interpret algorithmic knowledge, we rely on a knowledge algorithm for each agent in the system. We use the adversary's knowledge algorithm to capture the adversary's ability to draw conclusions from what he has seen. In this section, we show how we can capture different capabilities for the adversary in a natural way in this framework. We first show how to capture the standard model of adversary due to Dolev and Yao. We then show how to account for the adversary in the Duck-Duck-Goose protocol, and the adversary considered by Lowe [2002] that can perform self-validating guesses.

We start by considering passive (or eavesdropping) adversaries, which simply record every message exchanged by the principals; in Section 4.2, we consider active adversaries. For simplicity, we assume a single adversary per system; our results extend to the general case immediately, but the notation becomes cumbersome.

4.1 Passive Adversaries

Passive adversaries can be modeled formally as follows. An *interpreted algorithmic knowledge security system with passive adversary a* ($a \in \{1, \ldots, n\}$) is an interpreted algorithmic knowledge security system $\mathcal{I} = (\mathcal{R}, \pi, \mathtt{A}_1, \ldots, \mathtt{A}_n)$ satisfying the following constraints at all points (r, m):

P1. $r_a(m)$ consists only of $\mathsf{recv}(\mathsf{m})$ events.

P2. For all j and events $\mathsf{send}(j, \mathsf{m})$ in $r_j(m)$, there exists an event $\mathsf{recv}(\mathsf{m})$ in $r_a(m)$.

P1 captures the passivity of the adversary—he can only receive messages, not send any; P2 says that every message sent by a principal is copied to the adversary's local state. We next consider various knowledge algorithms for the adversary.

4.1.1 The Dolev-Yao Adversary

Consider the standard Dolev-Yao adversary [Dolev and Yao 1983]. This model is a combination of assumptions on the encryption scheme used and the capabilities of the adversaries. Specifically, the encryption scheme is seen as the free algebra generated by $\mathcal{P}$ and $\mathcal{K}$ over

operations $\cdot$ and $\{\!|\}$. Perhaps the easiest way to formalize this is to view the set $\mathcal{M}$ as the set of expressions generated by the grammar

$$\mathsf{m} ::= \mathsf{p} \mid \mathsf{k} \mid \{\!|\mathsf{m}|\!\}_{\mathsf{k}} \mid \mathsf{m} \cdot \mathsf{m}$$

(with $\mathsf{p} \in \mathcal{P}$ and $\mathsf{k} \in \mathcal{K}$). We then identify elements of $\mathcal{M}$ under the equivalence $\{\!|\{\!|\mathsf{m}|\!\}_{\mathsf{k}}|\!\}_{\mathsf{k}^{-1}} = \mathsf{m}$. We assume that there are no collisions; messages always have a unique decomposition. The only way that $\{\!|\mathsf{m}|\!\}_{\mathsf{k}} = \{\!|\mathsf{m}'|\!\}_{\mathsf{k}'}$ is if $\mathsf{m} = \mathsf{m}'$ and $\mathsf{k} = \mathsf{k}'$. We also make the standard assumption that concatenation and encryption have enough redundancy to recognize that a term is in fact a concatenation $\mathsf{m}_1 \cdot \mathsf{m}_2$ or an encryption $\{\!|\mathsf{m}|\!\}_{\mathsf{k}}$.

The Dolev-Yao model can be formalized by a relation $H \vdash_{DY} \mathsf{m}$ between a set H of messages and a message m. (Our formalization is equivalent to many other formalizations of Dolev-Yao in the literature, and is similar in spirit to that of Paulson [1998].) Intuitively, $H \vdash_{DY} \mathsf{m}$ means that an adversary can "extract" message m from a set of received messages and keys H, using the allowable operations. The derivation is defined using the following inference rules:

$$\frac{\mathsf{m} \in H}{H \vdash_{DY} \mathsf{m}} \quad \frac{H \vdash_{DY} \{\!|\mathsf{m}|\!\}_{\mathsf{k}} \quad H \vdash_{DY} \mathsf{k}^{-1}}{H \vdash_{DY} \mathsf{m}} \quad \frac{H \vdash_{DY} \mathsf{m}_1 \cdot \mathsf{m}_2}{H \vdash_{DY} \mathsf{m}_1} \quad \frac{H \vdash_{DY} \mathsf{m}_1 \cdot \mathsf{m}_2}{H \vdash_{DY} \mathsf{m}_2}.$$

In our framework, to capture the capabilities of a Dolev-Yao adversary, we specify how the adversary can tell if she in fact *has* a message, by defining a knowledge algorithm $\mathtt{A}_i^{\mathrm{DY}}$ for adversary i. Recall that a knowledge algorithm for agent i takes as input a formula and agent i's local state (which we are assuming contains the messages received by i). The most interesting case in the definition of $\mathtt{A}_i^{\mathrm{DY}}$ is when the formula is $\mathsf{has}_i(\mathsf{m})$. To compute $\mathtt{A}_i^{\mathrm{DY}}(\mathsf{has}_i(\mathsf{m}), \ell)$, the algorithm simply checks, for every message m' received by the adversary, whether m is a submessage of m', according to the keys that are known to the adversary. We assume that the adversary's initial state consists of the set of keys initially known by the adversary. This will typically contain, in a public-key cryptography setting, the public keys of all the agents. We use $initkeys(\ell)$ to denote the set of initial keys known by agent i in local state ℓ. (Recall that a local state for agent i is the sequence of events pertaining to agent i, including any initial information in the run, in this case, the keys initially known.) The function *submsg*, which can take apart messages created by concatenation, or decrypt messages as long as the adversary knows the decryption key, is used to check whether m is a submessage of m'. $\mathtt{A}_i^{\mathrm{DY}}(\mathsf{has}_i(\mathsf{m}), \ell)$ is defined as follows:

```
if m ∈ initkeys(ℓ) then return "Yes"
K = keysof(ℓ)
for each recv(m′) in ℓ
    if submsg(m, m′, K) then
        return "Yes"
return "No".
```

The auxiliary functions used by the algorithm are given in Figure 1.

According to the Dolev-Yao model, the adversary cannot explicitly compute anything interesting about what other messages agents have. Hence, for other primitives, including $\mathsf{has}_j(\mathsf{m})$ for $j \neq i$, $\mathtt{A}_i^{\mathrm{DY}}$ returns "?". For formulas of the form $K_j\varphi$ and $X_j\varphi$, $\mathtt{A}_i^{\mathrm{DY}}$ also

$submsg(\mathsf{m}, \mathsf{m}', K)$: if $\mathsf{m} = \mathsf{m}'$ then
    return *true*
if m' is $\{\!|\mathsf{m}_1|\!\}_{\mathsf{k}}$ and $\mathsf{k}^{-1} \in K$ then
    return $submsg(\mathsf{m}, \mathsf{m}_1, K)$
if m' is $\mathsf{m}_1 \cdot \mathsf{m}_2$ then
    return $submsg(\mathsf{m}, \mathsf{m}_1, K) \vee submsg(\mathsf{m}, \mathsf{m}_2, K)$
return *false*

$getkeys(\mathsf{m}, K)$: if $\mathsf{m} \in \mathcal{K}$ then
    return $\{\mathsf{m}\}$
if m' is $\{\!|\mathsf{m}_1|\!\}_{\mathsf{k}}$ and $\mathsf{k}^{-1} \in K$ then
    return $getkeys(\mathsf{m}_1, K)$
if m' is $\mathsf{m}_1 \cdot \mathsf{m}_2$ then
    return $getkeys(\mathsf{m}_1, K) \cup getkeys(\mathsf{m}_2, K)$
return $\{\}$

$keysof(\ell)$: $K \leftarrow initkeys(\ell)$
loop until no change in K
    $K \leftarrow \bigcup_{\mathsf{recv}(\mathsf{m}) \in \ell} getkeys(\mathsf{m}, K)$
return K

Figure 1: Dolev-Yao knowledge algorithm auxiliary functions

returns "?". For Boolean combinations of formulas, $\mathtt{A}_i^{\mathrm{DY}}$ returns the corresponding Boolean combination (where the negation of "?" is "?", the conjunction of "No" and "?" is "No", and the conjunction of "Yes" and "?" is "?") of the answer for each $\mathsf{has}_i(\mathsf{m})$ query.

The following result shows that an adversary using $\mathtt{A}_i^{\mathrm{DY}}$ recognizes (i.e., returns "Yes" to) $\mathsf{has}_i(\mathsf{m})$ in state ℓ if and only if m exactly the messages determined to be in the set of messages that can be derived (according to $\vdash_{DY}$) from the messages received in that state together with the keys initially known, Moreover, if a $\mathsf{has}_i(\mathsf{m})$ formula is derived at the point (r, m), then $\mathsf{has}_i(\mathsf{m})$ is actually true at (r, m) (so that $\mathtt{A}_i^{\mathrm{DY}}$ is sound).

Proposition 4.1. *Let $\mathcal{I} = (\mathcal{R}, \pi_{\mathcal{R}}^{\mathrm{S}}, \mathtt{A}_1, \ldots, \mathtt{A}_n)$ be an interpreted algorithmic knowledge security system where $\mathtt{A}_i = \mathtt{A}_i^{\mathrm{DY}}$. Then*

$$(\mathcal{I}, r, m) \models X_i(\mathsf{has}_i(\mathsf{m})) \text{ if and only if } \{\mathsf{m} : \mathsf{recv}(\mathsf{m}) \in r_i(m)\} \cup initkeys(\ell) \vdash_{DY} \mathsf{m}.$$

Moreover, if $(\mathcal{I}, r, m) \models X_i(\mathsf{has}_i(\mathsf{m}))$ then $(\mathcal{I}, r, m) \models \mathsf{has}_i(\mathsf{m})$.

Proof. Let $K = keysof(r_i(m))$. We must show that $\mathtt{A}_i^{\mathrm{DY}}(\mathsf{has}_i(\mathsf{m}), r_i(m)) =$ "Yes" if and only if $K \cup \{\mathsf{m} : \mathsf{recv}(\mathsf{m}) \in r_i(m)\} \vdash_{DY} \mathsf{m}$. It is immediate from the description of $\mathtt{A}_i^{\mathrm{DY}}$ and $\vdash_{DY}$ that this is true if $\mathsf{m} \in initkeys(r_i(m))$. If $\mathsf{m} \notin initkeys(r_i(m))$, then $\mathtt{A}_i^{\mathrm{DY}}(\mathsf{has}_i(\mathsf{m}), r_i(m)) =$ "Yes" if and only if $submsg(\mathsf{m}, \mathsf{m}', K) = true$ for some m' such that $\mathsf{recv}(\mathsf{m}') \in r_i(m)$. Next

observe that $submsg(\mathsf{m}, \mathsf{m}', K) = true$ if and only if $K \cup \{\mathsf{m}'\} \vdash_{DY} \mathsf{m}$: the "if" direction follows by a simple induction on the length of the derivation; the "only if" direction follows by a straightforward induction on the structure of m. Finally, observe that if M is a set of messages, then $K \cup M \vdash_{DY} \mathsf{m}$ if and only if $K \cup \{\mathsf{m}'\} \vdash_{DY} \mathsf{m}$ for some $\mathsf{m}' \in M$. The "if" direction is trivial. The "only if" direction follows by induction on the number of times the rule "from $\mathsf{m}' \in H$ infer $H \vdash_{DY} \mathsf{m}'$" is used to derive some $\mathsf{m}' \in M$. If it is never used, then it is easy to see that $K \vdash_{DY} \mathsf{m}'$. If it is used more than once, and the last occurrence is used to derive m', then it is easy to see that $K \cup \{\mathsf{m}'\} \vdash_{DY} \mathsf{m}'$ (the derivation just starts from the last use of this rule). The desired result is now immediate. ∎

In particular, if we have an interpreted algorithmic knowledge security system with a passive adversary a such that $\mathtt{A}_a = \mathtt{A}_a^{\mathrm{DY}}$, then Proposition 4.1 captures the knowledge of a passive Dolev-Yao adversary.

4.1.2 The Duck-Duck-Goose Adversary

The key advantage of our framework is that we can easily change the capabilities of the adversary beyond those prescribed by the Dolev-Yao model. For example, we can capture the fact that if the adversary knows the protocol, she can derive more information than she could otherwise. For instance, in the Duck-Duck-Goose example, assume that the adversary maintains in her local state a list of all the bits received corresponding to the key of the principal. We can easily write the algorithm so that if the adversary's local state contains all the bits of the key of the principal, then the adversary can decode messages that have been encrypted with that key. Specifically, assume that key k is being sent in the Duck-Duck-Goose example. Then for an adversary i, $\mathsf{has}_i(\mathsf{k})$ will be false until all the bits of the key have been received. This translates immediately into the following algorithm $\mathtt{A}_i^{\mathrm{DDG}}$:

if all the bits recorded in ℓ form k then
return "Yes" else return "No".

$\mathtt{A}_i^{\mathrm{DDG}}$ handles other formulas in the same way as $\mathtt{A}_i^{\mathrm{DY}}$.

Of course, nothing keeps us from combining algorithms, so that we can imagine an adversary intercepting both messages and key bits, and using an algorithm $\mathtt{A}_i$ that is a combination of the Dolev-Yao algorithm and the Duck-Duck-Goose algorithm; $\mathtt{A}_i(\varphi, \ell)$ is defined as follows:

if $\mathtt{A}_i^{\mathrm{DY}}(\varphi, \ell) =$ "Yes" then
    return "Yes"
else return $\mathtt{A}_i^{\mathrm{DDG}}(\varphi, \ell)$.

This assumes that the adversary knows the protocol, and hence knows when the key bits are being sent. The algorithm above captures this protocol-specific knowledge.

4.1.3 The Lowe Adversary

For a more realistic example of an adversary model that goes beyond Dolev-Yao, consider the following adversary model introduced by Lowe [2002] to analyze protocols subject to guessing attacks. The intuition is that some protocols provide for a way to "validate" the

guesses of an adversary. For a simple example of this, here is a simple challenge-based authentication protocol:

$$A \to S : A$$
$$S \to A : n_s$$
$$A \to S : \{\!|n_s|\!\}_{p_a}.$$

Intuitively, A tells the server S that she wants to authenticate herself. S replies with a challenge n_s. A sends back to S the challenge encrypted with her password p_a. Presumably, S knows the password, and can verify that she gets $\{\!|n_s|\!\}_{p_a}$. Unfortunately, an adversary can overhear both n_s and $\{\!|n_s|\!\}_{p_a}$, and can "guess" a value g for p_a and verify his guess by checking if $\{\!|n_s|\!\}_g = \{\!|n_s|\!\}_{p_a}$. The key feature of this kind of attack is that the guessing (and the validation) can be performed offline, based only on the intercepted messages.

To account for this capability of adversaries is actually fairly complicated. We present a slight variation of Lowe's description, mostly to make it notationally consistent with the rest of the section; we refer the reader to Lowe [2002] for a discussion of the design choices.

Lowe's model relies on a basic one-step reduction function, $S \rhd_l \mathsf{m}$, saying that the messages in S can be used to derive the message m. This is essentially the same as $\vdash_{DY}$, except that it represents a single step of derivation. Note that the derivation relation $\rhd_l$ is "tagged" by the kind of derivation performed (l).

$$\{\mathsf{m}, \mathsf{k}\} \rhd_{\mathsf{enc}} \{\!|\mathsf{m}|\!\}_{\mathsf{k}}$$
$$\{\{\!|\mathsf{m}|\!\}_{\mathsf{k}}, \mathsf{k}^{-1}\} \rhd_{\mathsf{dec}} \mathsf{m}$$
$$\{\mathsf{m}_1 \cdot \mathsf{m}_2\} \rhd_{\mathsf{fst}} \mathsf{m}_1$$
$$\{\mathsf{m}_1 \cdot \mathsf{m}_2\} \rhd_{\mathsf{snd}} \mathsf{m}_2.$$

Lowe also includes a reduction to derive $\mathsf{m}_1 \cdot \mathsf{m}_2$ from m_1 and m_2. We do not add this reduction to simplify the presentation. It is straightforward to extend our approach to deal with it.

Given a set H of message, and a sequence t of one-step reductions, we define inductively the set $[H]_t$ of messages obtained from the one-step reductions given in t:

$$[H]_{\langle\rangle} = H$$
$$[H]_{\langle S \rhd_l \mathsf{m} \rangle \cdot t} = \begin{cases} [H \cup \{\mathsf{m}\}]_t & \text{if } S \subseteq H \\ \textit{undefined} & \text{otherwise.} \end{cases}$$

Here, $\langle\rangle$ denotes the empty trace, and $t_1 \cdot t_2$ denotes trace concatenation. A trace t is said to be *monotone* if, intuitively, it does not perform any one-step reduction that "undoes" a previous one-step reduction. For example, the reduction $\{\mathsf{m}, \mathsf{k}\} \rhd \{\!|\mathsf{m}|\!\}_{\mathsf{k}}$ undoes the reduction $\{\{\!|\mathsf{m}|\!\}_{\mathsf{k}}, \mathsf{k}^{-1}\} \rhd \mathsf{m}$. (See Lowe [2002] for more details on undoing reductions.)

We say that a set H of messages *validates* a guess m if, intuitively, H contains enough information to verify that m is indeed a good guess. Intuitively, this happens if a value v (called a validator) can be derived from the messages in $H \cup \{\mathsf{m}\}$ in a way that uses the guess m, and either that (a) validator v can be derived in a different way from $H \cup \{\mathsf{m}\}$, (b) the validator v is already in $H \cup \{\mathsf{m}\}$, or (c) the validator v is a key whose inverse is derivable from $H \cup \{\mathsf{m}\}$. For example, in the protocol exchange at the beginning of this section, the

adversary sees the messages $H = \{n_s, \{\!|n_s|\!\}_{p_a}\}$, and we can check that H validates the guess $\mathsf{m} = p_a$: clearly, $\{n_s, \mathsf{m}\} \rhd_{\mathsf{enc}} \{\!|n_s|\!\}_{p_a}$, and $\{\!|n_s|\!\}_{p_a} \in H \cup \{\mathsf{m}\}$. In this case, the validator $\{\!|n_s|\!\}_{p_a}$ is already present in $H \cup \{\mathsf{m}\}$. For other examples of validation, we again refer to Lowe [2002].

We can now define the relation $H \vdash_L \mathsf{m}$ that says that m can be derived from H by a Lowe adversary. Intuitively, $H \vdash_L \mathsf{m}$ if m can be derived by Dolev-Yao reductions, or m can be guessed and validated by the adversary, and hence susceptible to an attack. Formally, $H \vdash_L \mathsf{m}$ if and only if $H \vdash_{DY} \mathsf{m}$ or there exists a monotone trace t, a set S, and a "validator" v such that

(1) $[H \cup \{\mathsf{m}\}]_t$ is defined;

(2) $S \rhd_l v$ is in t;

(3) there is no trace t' such that $S \subseteq [H]_{t'}$; and

(4) either:

  (a) there exists $(S', l') \neq (S, l)$ with $S' \rhd_{l'} v$ in t

  (b) $v \in H \cup \{\mathsf{m}\}$ or

  (c) $v \in \mathcal{K}$ and $v^{-1} \in [H \cup \{\mathsf{m}\}]_t$.

It is not hard to verify that this formalization captures the intuition about validation given earlier. Specifically, condition (1) says that the trace t is well-formed, condition (2) says that the validator v is derived from $H \cup \{\mathsf{m}\}$, condition (3) says that deriving the validator v depends on the guess m, and condition (4) specifies when a validator v validates a guess m, as given earlier.

We would now like to define a knowledge algorithm $\mathtt{A}_i^{\mathrm{L}}$ to capture the capabilities of the Lowe adversary. Again, the only case of real interest is what $\mathtt{A}_i^{\mathrm{L}}$ does on input $\mathsf{has}_i(\mathsf{m})$. $\mathtt{A}_i^{\mathrm{L}}(\mathsf{has}_i(\mathsf{m}), \ell)$ is defined as follows:

```
if A_i^DY(has_i(m), ℓ) = "Yes" then
    return "Yes"
if guess(m, ℓ) then
    return "Yes"
return "No".
```

The auxiliary functions used by the algorithm are given in Figure 2. (We have not concerned ourselves with matters of efficiency in the description of $\mathtt{A}_i^{\mathrm{L}}$; again, see Lowe [2002] for a discussion of implementation issues.)

As before, we can check the correctness and soundness of the algorithm:

Proposition 4.2. *Let $\mathcal{I} = (\mathcal{R}, \pi_{\mathcal{R}}^{\mathrm{S}}, \mathtt{A}_1, \ldots, \mathtt{A}_n)$ be an interpreted algorithmic knowledge security system where $\mathtt{A}_i = \mathtt{A}_i^{\mathrm{L}}$. Then*

$$(\mathcal{I}, r, m) \models X_i(\mathsf{has}_i(\mathsf{m})) \text{ if and only if } \{\mathsf{m} : \mathsf{recv}(\mathsf{m}) \in r_i(m)\} \cup \mathit{initkeys}(\ell) \vdash_L \mathsf{m}.$$

Moreover, if $(\mathcal{I}, r, m) \models X_i(\mathsf{has}_i(\mathsf{m}))$ then $(\mathcal{I}, r, m) \models \mathsf{has}_i(\mathsf{m})$.

$guess(\mathsf{m}, \ell)$: $H \leftarrow reduce(\{\mathsf{m} \ : \ \mathsf{recv}(\mathsf{m}) \text{ in } \ell\} \cup initkeys(\ell)) \cup \{\mathsf{m}\}$
$reds \leftarrow \{\}$
loop until $reductions(H) - reds$ is empty
  $(S, l, v) \leftarrow$ pick an element of $reductions(H) - reds$
  if $\exists (S', l', v) \in reds$ s.t. $S' \neq S$ and $l' \neq l$ then
    return "Yes"
  if $v \in H$ then
    return "Yes"
  if $v \in \mathcal{K}$ and $v^{-1} \in H$ then
    return "Yes"
  $reds \leftarrow reds \cup \{(S, l, v)\}$
  $H \leftarrow H \cup \{v\}$
return "No"

$reduce(H)$: loop until no change in H
  $r \leftarrow reductions(H)$
  for each (S, l, v) in r
    $H \leftarrow H \cup \{v\}$
return H

$reductions(H)$: $reds \leftarrow \{\}$
for each $\mathsf{m}_1 \cdot \mathsf{m}_2$ in H
  $reds \leftarrow \{(\{\mathsf{m}\}, \mathsf{fst}, \mathsf{m}_1), (\{\mathsf{m}\}, \mathsf{snd}, \mathsf{m}_2)\}$
for each $\mathsf{m}_1, \mathsf{m}_2$ in H
  if $\mathsf{m}_2 \in \mathcal{K}$ and $sub(\{\!|\mathsf{m}_1|\!\}_{\mathsf{m}_2}, H)$ then
    $reds \leftarrow \{(\{\mathsf{m}_1, \mathsf{m}_2\}, \mathsf{enc}, \{\!|\mathsf{m}_1|\!\}_{\mathsf{m}_2})\}$
  if m_1 is $\{\!|\mathsf{m}'|\!\}_{\mathsf{k}}$ and m_2 is k^{-1} then
    $reds \leftarrow \{(\{\mathsf{m}_1, \mathsf{m}_2\}, \mathsf{dec}, \mathsf{m}')\}$
return $reds$

$sub(\mathsf{m}, H)$: if $H = \{\mathsf{m}\}$ then
    return $true$
if $H = \{\mathsf{m}_1 \cdot \mathsf{m}_2\}$ then
    return $sub(\mathsf{m}, \{\mathsf{m}_1\}) \vee sub(\mathsf{m}, \{\mathsf{m}_2\})$
if $H = \{\{\!|\mathsf{m}'|\!\}_{\mathsf{k}}\}$ then
    return $sub(\mathsf{m}, \{\mathsf{m}'\})$
if $|H| > 1$ and $H = \{\mathsf{m}'\} \cup H'$ then
    return $sub(\mathsf{m}, \{\mathsf{m}'\}) \vee sub(\mathsf{m}, H')$
return $false$

Figure 2: Lowe knowledge algorithm auxiliary functions

Proof. Let $K = \mathit{keysof}(r_i(m))$. The proof is similar in spirit to that of Proposition 4.1, using the fact that if $\mathsf{m} \notin \mathit{initkeys}(r_i(m))$ and $K \cup \{\mathsf{m}' \mid \mathsf{recv}(\mathsf{m}') \in r_i(m)\} \not\vdash_{DY} \mathsf{m}$, then $\mathit{guess}(\mathsf{m}, r_i(m)) =$ "Yes" if and only if $K \cup \{\mathsf{m} \mid \mathsf{recv}(\mathsf{m}) \in r_i(m)\} \vdash_L \mathsf{m}$. The proof of this fact is essentially given by Lowe [2002], the algorithm $\mathtt{A}_i^{\mathrm{L}}$ being a direct translation of the CSP process implementing the Lowe adversary. Again, soundness with respect to $\mathsf{has}_i(m)$ follows easily. ∎

4.2 Active Adversaries

Up to now we have considered passive adversaries, which can intercept messages exchanged by protocol participants, but cannot actively participate in the protocol. Passive adversaries are often appropriate when the concern is confidentiality of messages. However, there are many attacks on security protocols that do not necessarily involve a breach of confidentiality. For instance, some authentication properties are concerned with ensuring that no adversary can pass himself off as another principal. This presumes that the adversary is able to interact with other principals. Even when it comes to confidentiality, there are clearly attacks that an active adversary can make that cannot be made by a passive adversary.

To analyze active adversaries, we need to consider what messages they can send. This, in turn depends on their capabilities, which we already have captured using knowledge algorithms. Formally, at a local state ℓ, an adversary using knowledge algorithm $\mathtt{A}_i$ can construct the messages in the set $C(\ell)$, defined to be the closure under *conc* and *encr* of the set $\{\mathsf{m} \mid \mathtt{A}_i(\mathsf{has}_i(\mathsf{m}), \ell) =$ "Yes"$\}$ of messages that adversary i has.

Once we consider active adversaries, we must consider whether they are insiders or outsiders. Intuitively, an insider is an adversary that other agents know about, and can initiate interactions with. (Insider adversaries are sometimes called *corrupt principals* or *dishonest principals*.) As we mentioned in the introduction, the difference between insiders and outsiders was highlighted by Lowe's [1995] man-in-the-middle attack of the Needham-Schroeder protocol.

An *interpreted algorithmic knowledge security system with active (insider) adversary a ($a \in \{1, \ldots, n\}$)* is an interpreted algorithmic knowledge security system $\mathcal{I} = (\mathcal{R}, \pi, \mathtt{A}_1, \ldots, \mathtt{A}_n)$ satisfying the following constraints at all points (r, m).

A1. For every $\mathsf{recv}(\mathsf{m}) \in r_a(m)$, there is a corresponding $\mathsf{send}(j, \mathsf{m})$ in $r_i(m)$ for some i.

A2. For every $\mathsf{send}(j, \mathsf{m}) \in r_a(m)$, we have $\mathsf{m} \in C(r_a(m))$.

A1 says that every message sent by the agents can be intercepted by the adversary and end up in the adversary's local state, rather than reaching its destination. A2 says that every message sent by the adversary must have been constructed out of the messages in his local state according to his capabilities. (Note that the adversary can forge the "send" field of the messages.)

To accommodate outsider adversaries, it suffices to add the restriction that no message is sent directly to the adversary. Formally, an *interpreted algorithmic knowledge security system with active (outsider) adversary a ($a \in \{1, \ldots, n\}$)* is an interpreted algorithmic knowledge security system $\mathcal{I} = (\mathcal{R}, \pi, \mathtt{A}_1, \ldots, \mathtt{A}_n)$ with an active insider adversary a such that for all points (r, m) and for all agents i, the following additional constraint is satisfied.

A3. For every $\mathsf{send}(j, \mathsf{m}) \in r_i(m)$, $j \neq a$.

5 Related Work

The issues we raise in this paper are certainly not new, and have been addressed, up to a point, in the literature. In this section, we review this literature, and discuss where we stand with respect to other approaches that have attempted to tackle some of the same problems.

As we mentioned in the introduction, the Dolev-Yao adversary is the most widespread adversary in the literature. Part of its attraction is its tractability, making it possible to develop formal systems to automatically check for safety with respect to such adversaries [Millen, Clark, and Freedman 1987; Mitchell, Mitchell, and Stern 1997; Paulson 1998; Lowe 1998; Meadows 1996]. The idea of moving beyond the Dolev-Yao adversary is not new. As we pointed out in Section 4.1.3, Lowe [2002] developed an adversary that can encode some amount of off-line guessing; we showed in Section 4.1.3 that we could capture such an adversary in our framework. Other approaches have the possibility of extending the adversary model. For instance, the framework of Paulson [1998], Clarke, Jha and Morrero [1998], and Lowe [1998] describe the adversary via a set of derivation rules, which could be modified by adding new derivation rules. We could certainly capture these adversaries by appropriately modifying our $\mathtt{A}_i^{\mathrm{DY}}$ knowledge algorithm. (Pucella [2006] studies the properties of algorithmic knowledge given by derivation rules in more depth.) However, these other approaches do not seem to have the flexibility of our approach in terms of capturing adversaries. Not all adversaries can be conveniently described in terms of derivation rules.

There are other approaches that weaken the Dolev-Yao adversary assumptions by either taking concrete encryption schemes into account, or at least adding new algebraic identities to the algebra of messages. Bieber [1990] does not assume that the encryption scheme is a free algebra, following an idea due to Merritt and Wolper [1985]. Even et al. [1985] analyze ping-pong protocols under RSA, taking the actual encryption scheme into account. The applied π-calculus of Abadi and Fournet [2001] permits the definition of an equational theory over the messages exchanged between processes, weakening some of the encryption scheme assumptions when the applied π-calculus is used to analyze security protocols. Since the encryption scheme used in our framework is a simple parameter to the logic, there is no difficulty in modifying our logic to reason about a particular encryption scheme, and hence we can capture these approaches in our framework. However, again, it seems that our approach is more flexible than these other approaches; not all adversaries can be defined simply by starting with a Dolev-Yao adversary and adding identities.

On a related note, the work of Abadi and Rogaway [2002], building on previous work by Bellare and Rogaway [1993], compare the results obtained by a Dolev-Yao adversary with those obtained by a more computational view of cryptography. They show that, under various conditions, the former is sound with respect to the latter, that is, terms that are assumed indistinguishable in the Dolev-Yao model remain indistinguishable under a concrete encryption scheme. It would be interesting to recast their analysis in our setting, which, as we argued, can capture both the Dolev-Yao adversary and more concrete adversaries.

The use of a logic based on knowledge or belief is also not new. A number of formal logics for analysis of security protocols that involve knowledge and belief have been introduced, going back to BAN logic [Burrows, Abadi, and Needham 1990], such as [Bieber 1990; Gong, Needham, and Yahalom 1990; Syverson 1990; Abadi and Tuttle 1991; Stubblebine

and Wright 1996; Wedel and Kessler 1996; Accorsi, Basin, and Viganò 2001]. The main problem with some of those approaches is that semantics of the logic (to the extent that one is provided) is typically not tied to protocol executions or attacks. As a result, protocols are analyzed in an idealized form, and this idealization is itself error-prone and difficult to formalize [Mao 1995].[5] While some of these approaches have a well-defined semantics and do not rely on idealization (e.g., [Bieber 1990; Accorsi, Basin, and Viganò 2001]), they are still restricted to (a version of) the Dolev-Yao adversary. In contrast, our framework goes beyond Dolev-Yao, as we have seen, and our semantics is directly tied to protocol execution. Other approaches have notions of knowledge that can be interpreted as a form of algorithmic knowledge ([Durgin, Mitchell, and Pavlovic 2003], for instance), but the interpretation of knowledge is fixed in the semantics of the logic.

The problem of logical omniscience in logics of knowledge is well known, and the literature describes numerous approaches to try to circumvent it. (See [Fagin, Halpern, Moses, and Vardi 1995, Chapter 10 and 11] for an overview.) In the context of security, this takes the form of using different semantics for knowledge, either by introducing *hiding* operators that hide part of the local state for the purpose of indistinguishability or by using notions such as *awareness* [Fagin and Halpern 1988] to capture an intruder's inability to decrypt [Accorsi, Basin, and Viganò 2001].[6] We now describe these two approaches in more detail.

The hiding approach is used in many knowledge-based frameworks as a way to define an essentially standard semantics for knowledge not subject to logical omniscience, at least as far as cryptography is concerned. Abadi and Tuttle [1991], for instance, map all messages that the agent cannot decrypt to a fixed symbol $\Box$; the semantics of knowledge is modified so that s and s' are indistinguishable to agent i when the local state of agent i in s and s' is the same after applying the mapping described above. Syverson and van Oorschot [1994] use a variant: rather than mapping all messages that an agent cannot decrypt to the same symbol $\Box$, they use a distinct symbol $\Box_x$ for each distinct term x of the free algebra modeling encrypted messages, and takes states containing these symbols to be indistinguishable if they are the same up to permutation of the set of symbols $\Box_x$. Thus, an adversary may still do comparisons of encrypted messages without attempting to decrypt them. Hutter and Schairer [2004] use this approach in their definition of information flow in the presence of symbolic cryptography, and Garcia et al. [2005] use it in their definition of anonymity in the presence of symbolic cryptography.[7] This approach deals with logical omniscience for encrypted messages: when the adversary receives a message m encrypted with a key that he does not know, the adversary does not know that he has m if there exists another state where he has received a different message m' encrypted with a key he does not know. However, the adversary can still perform arbitrary computations with the

[5] While more recent logical approaches (e.g., [Clarke, Jha, and Marrero 1998; Durgin, Mitchell, and Pavlovic 2003]) do not suffer from an idealization phase and are more tied to protocol execution, but they also do not attempt to capture knowledge and belief in any general way.

[6] A notion of algorithmic knowledge was defined by Moses [1988] and used by Halpern, Moses and Tuttle [1988] to analyze zero-knowledge protocols. Although related to algorithmic knowledge as defined here, Moses' approach does not use an explicit algorithm. Rather, it checks whether these exists an algorithm of a certain class (for example, a polynomial-time algorithm) that could compute such knowledge.

[7] A variant of this apprach is developed by Cohen and Dam [2005] to deal with logical omniscience in a first-order interpretation of BAN logic. Rather than using a symbol $\Box$ to model that a message is encrypted with an unknown key, they identify messages in different states encrypted using an unknown key.

data that he does know. Therefore, this approach does not directly capture *computational limitations*, something algorithmic knowledge takes into account.

Awareness is a more syntactical approach. Roughly speaking, the semantics for awareness can specify for every point a set of formulas of which an agent is aware. For instance, an agent may be aware of a formula without being aware of its subformulas. A general problem with awareness is determining the set of formulas of which an agent is aware at any point. One interpretation of algorithmic knowledge is that it characterizes what formulas an agent is aware of: those for which the algorithm says "Yes". In that sense, we subsume approaches based on awareness by providing them with an intuition. We should note that not every use of awareness in the security protocol analysis literature is motivated by the desire to model more general adversaries. Accorsi et al. [2001], for instance, describe a logic for reasoning about beliefs of agents participating in a protocol, much in the way that BAN logic is used to reason about beliefs of agents participating in a protocol. To deal with the logical omniscience problem, Accorsi et al. use awareness to restrict the set of facts that an agent can believe. Thus, an agent may be aware of which agent sent a message if she shares a secret with the sender of the message, and not be aware of that fact otherwise. This makes the thrust of their work different from ours.

6 Conclusion

We have presented a framework for security analysis using algorithmic knowledge. The knowledge algorithm can be tailored to account for both the capabilities of the adversary and the specifics of the protocol under consideration. Of course, it is always possible to take a security logic and extend it in an ad hoc way to reason about adversary with different capabilities. Our approach has a number of advantages over ad hoc approaches. In particular, it is quite general framework (we simply need to change the algorithm used by the adversary to change its capabilities, or add adversaries with different capabilities), and it permits reasoning about protocol-specific issues (for example, it can capture situations such as an agent sending the bits of her key).

Another advantage of our approach is that it naturally extends to the probabilistic setting. For instance, we can easily handle probabilistic protocols by considering multiagent systems with a probability distribution on the runs (see [1993]). We can also deal with knowledge algorithms that are probabilistic, although there are some additional subtleties that arise, since the semantics for X_i given here assumes that the knowledge algorithm is deterministic. In a companion paper [Halpern and Pucella 2005], we extend our approach to deal with probabilistic algorithmic knowledge, which lets us reason about a Dolev-Yao adversary that attempts to guess keys subject to a distribution. We hope to use this approach to to capture probabilistic adversaries of the kind studied by Lincoln et al. [1998].

The goal of this paper was to introduce a general framework for handling different adversary models in a natural way, not specifically to devise new attacks or adversary capabilities. With this framework, it should be possible to put on a formal foundation new attacks that are introduced by the community. We gave a concrete example of this with the "guess-and-confirm" attacks of Lowe [2002].

It is fair to ask at this point what we can gain by using this framework. For one thing, we believe that the ability of the framework to describe the capabilities of the adversary will

make it possible to specify the properties of security protocols more precisely. Of course, it may be the case that to prove correctness of a security protocol with respect to certain types of adversaries (for example, polynomial-time bounded adversaries) we will need to appeal to techniques developed in the cryptography community. However, we believe that it may well be possible to extend current model-checking techniques to handle more restricted adversaries (for example, Dolev-Yao extended with random guessing). This is a topic that deserves further investigation. In any case, having a logic where we can specify the abilities of adversaries is a necessary prerequisite to using model-checking techniques.

Acknowledgments

This research was inspired by discussions between the first author, Pat Lincoln, and John Mitchell, on a wonderful hike in the Dolomites. We also thank Sabina Petride for useful comments. Authors supported in part by NSF under grant CTC-0208535, by ONR under grants N00014-00-1-03-41 and N00014-01-10-511, by the DoD Multidisciplinary University Research Initiative (MURI) program administered by the ONR under grant N00014-01-1-0795, and by AFOSR under grant F49620-02-1-0101.

References

Abadi, M. and C. Fournet (2001). Mobile values, new names, and secure communication. In *Proc. 28th Annual ACM Symposium on Principles of Programming Languages (POPL'01)*, pp. 104–115.

Abadi, M. and P. Rogaway (2002). Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology 15*(2), 103–127.

Abadi, M. and M. R. Tuttle (1991). A semantics for a logic of authentication. In *Proc. 10th ACM Symposium on Principles of Distributed Computing (PODC'91)*, pp. 201–216.

Accorsi, R., D. Basin, and L. Viganò (2001). Towards an awareness-based semantics for security protocol analysis. In J. Goubault-Larrecq (Ed.), *Proc. Workshop on Logical Aspects of Cryptographic Protocol Verification*, Volume 55.1 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers.

Bellare, M. and P. Rogaway (1993). Entity authentication and key distribution. In *Proc. 13th Annual International Cryptology Conference (CRYPTO'93)*, Volume 773 of *Lecture Notes in Computer Science*, pp. 232–249. Springer-Verlag.

Bieber, P. (1990). A logic of communication in hostile environment. In *Proc. 3rd IEEE Computer Security Foundations Workshop (CSFW'90)*, pp. 14–22. IEEE Computer Society Press.

Burrows, M., M. Abadi, and R. Needham (1990). A logic of authentication. *ACM Transactions on Computer Systems 8*(1), 18–36.

Clarke, E., S. Jha, and W. Marrero (1998). Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*.

Cohen, M. and M. Dam (2005). Logical omniscience in the semantics of BAN logic. In *Proc. Workshop on Foundations of Computer Security (FCS'05)*.

Dolev, D. and A. C. Yao (1983). On the security of public key protocols. *IEEE Transactions on Information Theory 29*(2), 198–208.

Durgin, N. A., J. C. Mitchell, and D. Pavlovic (2003). A compositional logic for proving security properties of protocols. *Journal of Computer Security 11*(4), 677–722.

Even, S., O. Goldreich, and A. Shamir (1985). On the security of ping-pong protocols when implemented using the RSA. In *Proc. Conference on Advances in Cryptology (CRYPTO'85)*, Volume 218 of *Lecture Notes in Computer Science*, pp. 58–72. Springer-Verlag.

Fagin, R. and J. Y. Halpern (1988). Belief, awareness, and limited reasoning. *Artificial Intelligence 34*, 39–76.

Fagin, R. and J. Y. Halpern (1994). Reasoning about knowledge and probability. *Journal of the ACM 41*(2), 340–367.

Fagin, R., J. Y. Halpern, Y. Moses, and M. Y. Vardi (1995). *Reasoning about Knowledge*. MIT Press.

Garcia, F. D., I. Hasuo, W. Pieters, and P. van Rossum (2005). Provable anonymity. In *Proc. 3rd ACM Workshop on Formal Methods in Security Engineering (FMSE 2005)*, pp. 63–72. ACM Press.

Gong, L., R. Needham, and R. Yahalom (1990). Reasoning about belief in cryptographic protocols. In *Proc. 1990 IEEE Symposium on Security and Privacy*, pp. 234–248. IEEE Computer Society Press.

Gordon, A. D. and A. Jeffrey (2003). Authenticity by typing for security protocols. *Journal of Computer Security 11*(4), 451–520.

Halpern, J. Y., Y. Moses, and M. R. Tuttle (1988). A knowledge-based analysis of zero knowledge. In *Proc. 20th Annual ACM Symposium on the Theory of Computing (STOC'88)*, pp. 132–147.

Halpern, J. Y. and K. O'Neill (2002). Secrecy in multiagent systems. In *Proc. 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, pp. 32–46. IEEE Computer Society Press.

Halpern, J. Y. and R. Pucella (2003). On the relationship between strand spaces and multi-agent systems. *ACM Transactions on Information and System Security 6*(1), 43–70.

Halpern, J. Y. and R. Pucella (2005). Probabilistic algorithmic knowledge. *Logical Methods in Computer Science 1*(3:1).

Halpern, J. Y. and M. R. Tuttle (1993). Knowledge, probability, and adversaries. *Journal of the ACM 40*(4), 917–962.

Hutter, D. and A. Schairer (2004). Possibilistic information flow control in the presence of encrypted communication. In *Proc. 9th European Symposium on Research in Computer Security (ESORICS'04)*, Volume 3193 of *Lecture Notes in Computer Science*, pp. 209–224. Springer-Verlag.

Kripke, S. (1963). A semantical analysis of modal logic I: normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 9*, 67–96.

Lincoln, P., J. C. Mitchell, M. Mitchell, and A. Scedrov (1998). A probabilistic polytime framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security (CCS'98)*, pp. 112–121.

Lowe, G. (1995). An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters 56*, 131–133.

Lowe, G. (1998). Casper: A compiler for the analysis of security protocols. *Journal of Computer Security 6*, 53–84.

Lowe, G. (2002). Analysing protocols subject to guessing attacks. In *Proc. Workshop on Issues in the Theory of Security (WITS'02)*.

Mao, W. (1995). An augmentation of BAN-like logics. In *Proc. 8th IEEE Computer Security Foundations Workshop (CSFW'95)*, pp. 44–56. IEEE Computer Society Press.

Meadows, C. (1996). The NRL protocol analyzer: An overview. *Journal of Logic Programming 26*(2), 113–131.

Merritt, M. and P. Wolper (1985). States of knowledge in cryptographic protocols. Unpublished manuscript.

Millen, J. K., S. C. Clark, and S. B. Freedman (1987). The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering 13*(2), 274–288.

Mitchell, J., M. Mitchell, and U. Stern (1997). Automated analysis of cryptographic protocols using Murφ. In *Proc. 1997 IEEE Symposium on Security and Privacy*, pp. 141–151. IEEE Computer Society Press.

Moore, J. H. (1988). Protocol failures in cryptosystems. *Proceedings of the IEEE 76*(5), 594–602.

Moses, Y. (1988). Resource-bounded knowledge. In *Proc. 2nd Conference on Theoretical Aspects of Reasoning about Knowledge (TARK'88)*, pp. 261–276. Morgan Kaufmann.

Needham, R. M. and M. D. Schroeder (1978). Using encryption for authentication in large networks of computers. *Communications of the ACM 21*(12), 993–999.

Paulson, L. C. (1998). The inductive approach to verifying cryptographic protocols. *Journal of Computer Security 6*(1/2), 85–128.

Pucella, R. (2006). Deductive algorithmic knowledge. *Journal of Logic and Computation 16*(2), 287–309.

Ryan, P. Y. A. and S. A. Schneider (1998). An attack on a recursive authentication protocol: A cautionary tale. *Information Processing Letters 65*(1), 7–10.

Stubblebine, S. and R. Wright (1996). An authentication logic supporting synchronization, revocation, and recency. In *Proc. 3rd ACM Conference on Computer and Communications Security (CCS'96)*. ACM Press.

Syverson, P. (1990). A logic for the analysis of cryptographic protocols. NRL Report 9305, Naval Research Laboratory.

Syverson, P. and I. Cervesato (2001). The logic of authentication protocols. In *Proc. 1st International School on Foundations of Security Analysis and Design (FOSAD'00)*, Volume 2171 of *Lecture Notes in Computer Science*, pp. 63–137.

Syverson, P. F. and P. C. van Oorschot (1994). On unifying some cryptographic protocol logics. In *Proc. 1994 IEEE Symposium on Security and Privacy*, pp. 14–28. IEEE Computer Society Press.

Thayer, F. J., J. C. Herzog, and J. D. Guttman (1999). Strand spaces: Proving security protocols correct. *Journal of Computer Security 7*(2/3), 191–230.

Wedel, G. and V. Kessler (1996). Formal semantics for authentication logics. In *Proc. 4th European Symposium on Research in Computer Security (ESORICS'96)*, Volume 1146 of *Lecture Notes in Computer Science*, pp. 219–241. Springer-Verlag.